Nasdaq Calypso

Data Uploader Developer's Guide

Version 17 - 18

Revision 5.0
February 2022
Approved

Document History

Revision	Published	Summary of Changes
1.0	March 2015	First edition.
2.0	March 2019	Added important upgrade information.
3.0	November 2020	Added information about Web Services.
4.0	December 2020	Updated for version 9.3.6. Introduction of calypsoServicesInterfacesMs.bat for Data Uploader REST API.
5.0	February 2022	As of version 17.0, REST APIs documentation is provided under the REST APIs page on the Documentation Portal and under <calypso home>/docs.

This document explains how to extend the Calypso Data Uploader framework.

Table of Contents

Introduction

This document explains how to extend the Calypso Data Uploader framework. It provides the different extension points available in the framework and gives examples on how to extend and customize them.

The Data Uploader currently supports different Calypso objects to be uploaded into the system, like Trades (different product types), Corporate Actions, Books and Legal Entities. This document explains the different components which are involved in the Data Uploader framework and how to extend them for adding new product types for trades or any other Calypso object.

[Note: This guide is to be used by clients licensed for direct use of the Calypso Data Uploader interface only.

This document lists all APIs available for clients to customize. Only these documented APIs should be extended by clients. Any other functions, even if they are public, are not supported for client extensions and may be changed by Calypso without notice.

Calypso reserves the right to deprecate documented APIs and provide alternate features offering similar functionality wherever appropriate.]

Important Upgrade Information

2.1 Deprecations

The BOMessage mode (workflow-based mode) will be deprecated in the upcoming 5.0.0 version of the Data Uploader in Q1 or Q2 of 2016.

All the clients are requested to upgrade to the Local mode with persistent messages for All/None/Failure messages.

2.2 Data Uploader XSD

The Data Uploader XSD is internal to the Data Uploader and cannot be used as an API. Between major versions, we cannot guarantee backward compatibility.

For example, in 7.0.0 onwards we have changed the "trade id" related tags from type "int" to "long" and when migrating to version 7.0.0, you need to make sure that your custom code is changed accordingly.

Uniqueness of objects in Data Uploader

The Data Uploader framework allows uploading new objects and handling life cycle actions (NEW/AMEND/DELETE/CANCEL/TERMINATE/NOVATE). In order to do that, the ability to identify an object uniquely is very important.

Data Uploader does not use the unique id generated by calypso for uniqueness, as the data in most scenarios come from an external system and the internal id generated by calypso may not be known, also the source system has its own id.

To achieve this goal, we use the following.

Object Type	Unique Constraint
Trade	External Reference (SOURCE_PO_DealID) where DealID is the ID in the Source System.
Product	Sec Codes (CUSIP/ISIN/TICKER)
Static Data	Name

Similarly, other objects have support for Ext Reference, like Legal Entities. In such cases, the Ext Reference must be unique (even if Calypso application does not enforce it). The uniqueness is a requirement to use Calypso Data Uploader for integration.

It is important not to change this logic.

APIs

The Data Uploader framework exposes certain APIs (methods in a Utility class) to help the internal and implementation teams integrate with the Data Uploader module. These APIs expect an XML string or the JAXB representation of the XML, and depending on the API used, the object represented by the XML or the JAXB object passed is validated and uploaded into Calypso, and the Acknowledgement object is returned to the caller of the API.

The Acknowledgement object returned to the caller of the API is a Java object, representing the number of objects passed to the API, the number of objects uploaded successfully, the the number of objects that failed to upload, and the errors when the upload failed.

The APIs are designed to include all the features supported by the Data Uploader module such as mapping/extensions.

The APIs are designed to execute at the client side or the server side. When executed on the client side, the APIs make use of Calypso BOCache / LocalCache, and perform well. The server side APIs makes use of Calypso BO Messages, and the objects are validated and uploaded using the message workflows.

In Process a.k.a. Local API

These are APIs that can be used as a library and embedded in your custom application. The custom application needs connectivity to Calypso. They are called LocalAPIs since all the processing is done in-process i.e. within the process space of the client application.

All Local APIs are present in "`com.calyso.tk.util.DataUploaderUtil.java`".

4.1 Local API 1: Upload Object via JAXB

```
	/**
	* This API, given a CalypsoUploadDocument, validates each object present in the document and uploads if there are no validation errors
	 * and return the acknowledgement object. Only the objects with no validation errors are uploaded.
	 * @param calypsoUploadDocument
	* @return
	* @throws Exception
	*/
	public static CalypsoAcknowledgement uploadCalypsoObject(CalypsoUploadDocument calypsoUploadDocument) throws Exception {
	}
```

Description:

This API expects a Calypso Upload Document object to be passed; this document may contain many or one object at least, if there are multiple objects, and they are not related to each other then they will be multi-threaded, and once the processing (validation/uploader) of all the objects is completed, the acknowledgement is generated and returned back.

NO BO Message & and Task station integration.

4.2 Local API 2: Upload using XML

```
    /**
     * This API, takes xml string and converts to CalypsoUploadDocument and uses the API
uploadCalypsoObject(CalypsoUploadDocument)
     * @param xmlmessage
     * @return
     * @throws Exception
     */
    public static CalypsoAcknowledgement uploadXML(String xmlmessage) throws Exception {
    }
```

Description:

This API is similar to the API 1, except that it accepts an XML string, which is converted to a Calypso Upload Document Object and invokes API 1.

Please note that the passed XML must be an uploader XML with root tag `<CalypsoUploadDocument>`.

4.3 Local API 3: Create In memory Object from JAXB

```
/**This API will validate the Calypso Object passed in the CalypsoUploadDocument and create an in memory upload
object
       * if there are no validation errors, and return the acknowledgement as well as the upload object
encapsulated in UploadResult
       * Note that the caller has to pass one object per document, only the first Object is taken and processed
and the
      * result for the same is returned*/
      public static UploadResult createCalypsoObject(CalypsoUploadDocument uploadDocument) {}
```

Description:

This API has a different behavior compared to the rest of the APIs; it does exactly everything that all other APIs do with respect to validating/mapping/customizing but when it comes to loading, it does not save the object into Calypso. Instead, an in-memory object is created and returned back to the user along with the acknowledgement, encapsulated in the UploadResult object.

NO BO Message & and Task station integration.

Example:

```
	UploadResult result = DataUploaderUtil.createCalypsoObject(document);
	//if the document contains trade xml jaxb then the object created would be trade. For every type of object passed, the respective calypso object will be created.
	if (result != null) {
		Object uploadObject=result.getUploadObject();
		CalypsoAcknowledgement ack = result.getCalypsoAcknowledgement ();
		if (uploadObject != null && uploadObject instanceof Trade) {
		{
			Trade trade = (Trade) uploadObject;
			//trade can be saved.
		}
	}
```

How to Use Local API

5.1 Writing a Java Application that calls the Local API

The Local API can be called from any Calypso application or engine; the following example explains how to write java code to call the local API.

```
public class TestLocalAPI {


	static Unmarshaller unmarshaller = null;
	static JAXBContext jc = null;
	static {
		try {
			jc = JAXBContext.newInstance("com.calypso.tk.upload.jaxb");
			unmarshaller = jc.createUnmarshaller();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
	public static void main(String[] args) throws Exception {
		ConnectionUtil.connect("calypso_user", "calypso", "TestLocalAPI", "rel1300");
		File file = new File("c:\\tmp\\CUD37507.xml");
		CalypsoUploadDocument calypsoDoc = (CalypsoUploadDocument)unmarshaller.unmarshal(file);
		try {
			CalypsoAcknowledgement ack = DataUploaderUtil.uploadCalypsoObject(calypsoDoc);
			String output = DataUploaderUtil.marshallAcknowledgement(ack);
			System.out.println(output);
		} catch (Exception e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
}
```

5.2 Using File Watcher

The File watcher has the following properties in the file "`datauploader.properties`".

```
uploadMode=Local

persistMessages=All/Failure/None
```

Please refer to the next section for more details.

Data Uploader Web Services

Data Uploader supports web-services (RESTful API) from Calypso 15.2.0.0 onwards.

You can connect to this web-service and upload object(s) into Calypso in CalypsoUploadDocument format. These web-services support both XML and JSON format. The supported JSON formats are converted directly from Data Uploader XML files.

As of version 17.0, REST APIs documentation is provided under the REST APIs page on the Documentation Portal and under `<calypso home>/docs`.

In version 15.2, the web-services are deployed into the Engine server.

In version 16.1.0.55 onwards, the web-services are deployed into the calypsoServices. You can start the calypsoServices server using calypsoServices.bat/.sh.

As of 16.1.0.65, it is deployed into the calypsoServicesInterfacesMs. You can start the calypsoServicesInterfacesMs server using calypsoServicesInterfacesMs.bat/.sh.

Below are the paths to the API.

15.2 - Base URL= http://<host>:<port>/engineserver/services/

v1 – From 16.1.0.55 - Base URL = http://<host>:<port>/datauploader-service/services/

v2 – From 16.1.0.94 - Base URL = http://<host>:<port>/datauploader-service/

The default port of calypsoServices server is 8801.

The default port of calypsoServicesInterfacesMs server is 8801.

End User access permissions are disabled by default. You can enable access permissions using environment property DISABLE_END_USER_PERMISSIONING=false. When access permissions are enabled, the user needs the access permission function saveTrade to save imported trades.

You can also use the JVM argument -DendUserPermissioning=true for calypsoServiceInterfaceMs.

6.1 REST API Version 1

HTTP Method	URL	Input Format Supported	Output Format Supported	Service Type
POST	datauploader/v1/upload Optional query parameters: • persistMessage [Valid values: None, Failure, All; Default: None] • ignoreWarnings [Format: false, false]	XML / JSON	XML / JSON	Synchronous
POST	datauploader/v1/uploadAll	XML / JSON	XML / JSON Request Id	Asynchronous
GET	datauploader/v1/acknowledgement?requestId=?	Request Id	XML/JSON	Synchronous

6.2 REST API Version 2

6.2.1 HTTP Verbs

The following are the general HTTP verbs to use across all invocations.

Verb	Usage
GET	Used to retrieve a resource
POST	Used to create a new resource
PATCH	Used to update an existing resource, including partial updates
PUT	Used to update an existing resource, full updates only
DELETE	Used to delete an existing resource

6.2.2 HTTP Status Codes

The following are the supported status codes

Status code	Usage
200 OK	Standard response for successful HTTP requests. The actual response will depend on the request method used. In a GET request, the response will contain an entity corresponding to the requested resource. In a POST request, the response will contain an entity describing or containing the result of the action.
201 Created	The request has been fulfilled and resulted in a new resource being created.
204 No Content	The server successfully processed the request but is not returning any content.
400 Bad Request	The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).
404 Not Found	The requested resource could not be found but may be available again in the future. Subsequent requests by the client are permissible.

6.2.3 Upload Interface Message as String

A `POST` uploads the Interface message. Returns Acknowledgement with status and error messages if any.

Request Parameters

Parameter	Description
uploadSource	optional: "Source Interface Name. Default value is Uploader"
uploadFormat	optional: "Interface format. Default value is UploaderXML"
persistMessage	optional: "Valid values are All,Failure,None. Default value is None."
ignoreWarnings	optional: "Valid values are true/false. Default value is true"

Request Example

```
POST /api/v2/datauploader/upload-
message?uploadSource=Uploader&uploadFormat=UploaderXML&persistMessage=None&ignoreWarnings=true HTTP/1.1

Content-Type: text/plain

Host: localhost:8080
```

```
Content-Length: 5269


<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CalypsoUploadDocument>
    <CalypsoTrade tradeType="BasisSwap">
        <ExternalReference>AveragingTrade_1</ExternalReference>
        <TradeId>0</TradeId>
        <Action>NEW</Action>
        <TradeCounterParty>CP</TradeCounterParty>
               <CounterPartyRole>CounterParty</CounterPartyRole>
        <TradeBook>Global</TradeBook>
        <TradeCurrency>USD</TradeCurrency>
        <TradeNotional>0.0</TradeNotional>
        <TradeDateTime>2012-05-11T18:38:48Z</TradeDateTime>
        <TradeSettleDate>2012-05-15</TradeSettleDate>
        <StartDate>2012-05-15</StartDate>
        <MaturityDate>2013-05-15</MaturityDate>
        <TraderName>trader20</TraderName>
        <SalesPerson>NONE</SalesPerson>
        <Comment></Comment>
        <ProductType>InterestRateSwap</ProductType>
        <ProductSubType>BasisSwap</ProductSubType>
        <Product>
            <InterestRateSwap>
                <FxRate/>
                <MarkToMarket/>
                <IndexResetDate/>
                <MToMAdjFirst/>
                <SwapLeg>
                    <LegType>Float</LegType>
                    <PayRec>REC</PayRec>
                    <Currency>USD</Currency>
                    <Amount>2.0E7</Amount>
                    <StartDate>2012-05-15</StartDate>
                    <EndDate>2013-05-15T23:59:00.000-04:00</EndDate>
                    <Rate/>
                    <RateIndex>FEDFUNDS</RateIndex>
```

```
            <RateIndexSource>FEDFUNDS1</RateIndexSource>
            <Spread>0.5</Spread>
            <Tenor>1D</Tenor>
            <StubPeriod>NONE</StubPeriod>
            <SpecificFirstDate/>
            <SpecificLastDate/>
            <FirstReset>NONE</FirstReset>
            <FirstRate/>
            <HolidayCode>
                <Holiday>LON</Holiday>
                <Holiday>NYC</Holiday>
            </HolidayCode>
            <InterestCompounding/>
            <InterestCompoundingMethod/>
            <InterestCompoundingFrequency>QTR</InterestCompoundingFrequency>
            <ResetFrequency>DLY</ResetFrequency>
            <ResetMethod>Weighted</ResetMethod>
            <FloatingRateReset>true</FloatingRateReset>
            <ResetLag>0</ResetLag>
            <PaymentFrequency>QTR</PaymentFrequency>
            <DateRollConvention>MOD_FOLLOW</DateRollConvention>
            <AccrualPeriodAdjustment>ADJUSTED</AccrualPeriodAdjustment>
            <RollDay>15</RollDay>
            <PaymentLag>0</PaymentLag>
            <DayCountConvention>ACT/360</DayCountConvention>
            <CouponOffsetBusDayB>true</CouponOffsetBusDayB>
            <ResetOffsetBusDayB>true</ResetOffsetBusDayB>
            <ResetHolidays>
                <Holiday>NYC</Holiday>
            </ResetHolidays>
            <CouponPaymentAtEnd>true</CouponPaymentAtEnd>
        </SwapLeg>
        <SwapLeg>
            <LegType>Float</LegType>
            <PayRec>PAY</PayRec>
            <Currency>USD</Currency>
            <Amount>2.0E7</Amount>
            <StartDate>2012-05-15</StartDate>
```

```
                <EndDate>2013-05-15T23:59:00.000-04:00</EndDate>
                <Rate/>
                <RateIndex>LIBOR</RateIndex>
                <RateIndexSource>LIBOR01</RateIndexSource>
                <Spread>0.2</Spread>
                <Tenor>3M</Tenor>
                <StubPeriod>NONE</StubPeriod>
                <SpecificFirstDate/>
                <SpecificLastDate/>
                <FirstReset>NONE</FirstReset>
                <FirstRate/>
                <HolidayCode>
                    <Holiday>LON</Holiday>
                    <Holiday>NYC</Holiday>
                </HolidayCode>
                <InterestCompounding/>
                <InterestCompoundingMethod/>
                <InterestCompoundingFrequency>QTR</InterestCompoundingFrequency>
                <ResetFrequency/>
                <ResetMethod/>
                <FloatingRateReset/>
                <ResetLag>-2</ResetLag>
                <PaymentFrequency>QTR</PaymentFrequency>
                <DateRollConvention>MOD_FOLLOW</DateRollConvention>
                <AccrualPeriodAdjustment>ADJUSTED</AccrualPeriodAdjustment>
                <RollDay>15</RollDay>
                <PaymentLag>0</PaymentLag>
                <DayCountConvention>ACT/360</DayCountConvention>
                <CouponOffsetBusDayB>true</CouponOffsetBusDayB>
                <ResetOffsetBusDayB>true</ResetOffsetBusDayB>
                <ResetHolidays>
                    <Holiday>LON</Holiday>
                </ResetHolidays>
                <ResetTiming>BEG_PER</ResetTiming>
                <CouponPaymentAtEnd>true</CouponPaymentAtEnd>
            </SwapLeg>
            <Discounted>false</Discounted>
        </InterestRateSwap>
```

```
        </Product>
     </CalypsoTrade>
</CalypsoUploadDocument>
```

Response Example

```
HTTP/1.1 200 OK
calcorrelationid: 5e8363b0-0128-4cc5-889a-b7aa60c46431#junit/084f629f
nodeid:
Content-Type: application/json;charset=UTF-8
Content-Length: 641


<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CalypsoAcknowledgement UploadFile="IRS_Averaging.xml" UploadDate="Mon Aug 24 13:27:08 IDT 2020"
TotalTime="1407" Uploaded="1" Rejected="0" Received="1">
    <CalypsoTrades Uploaded="1" Rejected="0" Received="1">
        <CalypsoTrade>
            <ExternalRef>AveragingTrade_1</ExternalRef>
            <ProcessingOrg>GLOBAL BALANCED FUND</ProcessingOrg>
            <CalypsoTradeId>1135711</CalypsoTradeId>
            <Status>Success</Status>
            <Action>NEW</Action>
            <TradeCustomKeywords/>
        </CalypsoTrade>
    </CalypsoTrades>
</CalypsoAcknowledgement>
```

6.2.4 Upload Message as XML

A `POST` uploads the xml data. Returns Acknowledgement with status and error messages if any.

Request Parameters

Parameter	Description
persistMessage	optional: "Valid values are All,Failure,None. Default value is None."
ignoreWarnings	optional: "Valid values are true/false. Default value is true"

Request Example

```
POST /api/v2/datauploader/upload?persistMessage=None&ignoreWarnings=true HTTP/1.1
Content-Type: application/xml
Host: localhost:8080
Content-Length: 5269

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CalypsoUploadDocument>
    <CalypsoTrade tradeType="BasisSwap">
        <ExternalReference>AveragingTrade_1</ExternalReference>
        <TradeId>0</TradeId>
        <Action>NEW</Action>
        <TradeCounterParty>CP</TradeCounterParty>
               <CounterPartyRole>CounterParty</CounterPartyRole>
        <TradeBook>Global</TradeBook>
        <TradeCurrency>USD</TradeCurrency>
        <TradeNotional>0.0</TradeNotional>
        <TradeDateTime>2012-05-11T18:38:48Z</TradeDateTime>
        <TradeSettleDate>2012-05-15</TradeSettleDate>
        <StartDate>2012-05-15</StartDate>
        <MaturityDate>2013-05-15</MaturityDate>
        <TraderName>trader20</TraderName>
        <SalesPerson>NONE</SalesPerson>
        <Comment></Comment>
        <ProductType>InterestRateSwap</ProductType>
        <ProductSubType>BasisSwap</ProductSubType>
        <Product>
            <InterestRateSwap>
                <FxRate/>
                <MarkToMarket/>
                <IndexResetDate/>
                <MToMAdjFirst/>
                <SwapLeg>
                    <LegType>Float</LegType>
                    <PayRec>REC</PayRec>
                    <Currency>USD</Currency>
                    <Amount>2.0E7</Amount>
```

```
            <StartDate>2012-05-15</StartDate>
            <EndDate>2013-05-15T23:59:00.000-04:00</EndDate>
            <Rate/>
            <RateIndex>FEDFUNDS</RateIndex>
            <RateIndexSource>FEDFUNDS1</RateIndexSource>
            <Spread>0.5</Spread>
            <Tenor>1D</Tenor>
            <StubPeriod>NONE</StubPeriod>
            <SpecificFirstDate/>
            <SpecificLastDate/>
            <FirstReset>NONE</FirstReset>
            <FirstRate/>
            <HolidayCode>
                <Holiday>LON</Holiday>
                <Holiday>NYC</Holiday>
            </HolidayCode>
            <InterestCompounding/>
            <InterestCompoundingMethod/>
            <InterestCompoundingFrequency>QTR</InterestCompoundingFrequency>
            <ResetFrequency>DLY</ResetFrequency>
            <ResetMethod>Weighted</ResetMethod>
            <FloatingRateReset>true</FloatingRateReset>
            <ResetLag>0</ResetLag>
            <PaymentFrequency>QTR</PaymentFrequency>
            <DateRollConvention>MOD_FOLLOW</DateRollConvention>
            <AccrualPeriodAdjustment>ADJUSTED</AccrualPeriodAdjustment>
            <RollDay>15</RollDay>
            <PaymentLag>0</PaymentLag>
            <DayCountConvention>ACT/360</DayCountConvention>
            <CouponOffsetBusDayB>true</CouponOffsetBusDayB>
            <ResetOffsetBusDayB>true</ResetOffsetBusDayB>
            <ResetHolidays>
                <Holiday>NYC</Holiday>
            </ResetHolidays>
            <CouponPaymentAtEnd>true</CouponPaymentAtEnd>
        </SwapLeg>
        <SwapLeg>
            <LegType>Float</LegType>
```

```
<PayRec>PAY</PayRec>
<Currency>USD</Currency>
<Amount>2.0E7</Amount>
<StartDate>2012-05-15</StartDate>
<EndDate>2013-05-15T23:59:00.000-04:00</EndDate>
<Rate/>
<RateIndex>LIBOR</RateIndex>
<RateIndexSource>LIBOR01</RateIndexSource>
<Spread>0.2</Spread>
<Tenor>3M</Tenor>
<StubPeriod>NONE</StubPeriod>
<SpecificFirstDate/>
<SpecificLastDate/>
<FirstReset>NONE</FirstReset>
<FirstRate/>
<HolidayCode>
    <Holiday>LON</Holiday>
    <Holiday>NYC</Holiday>
</HolidayCode>
<InterestCompounding/>
<InterestCompoundingMethod/>
<InterestCompoundingFrequency>QTR</InterestCompoundingFrequency>
<ResetFrequency/>
<ResetMethod/>
<FloatingRateReset/>
<ResetLag>-2</ResetLag>
<PaymentFrequency>QTR</PaymentFrequency>
<DateRollConvention>MOD_FOLLOW</DateRollConvention>
<AccrualPeriodAdjustment>ADJUSTED</AccrualPeriodAdjustment>
<RollDay>15</RollDay>
<PaymentLag>0</PaymentLag>
<DayCountConvention>ACT/360</DayCountConvention>
<CouponOffsetBusDayB>true</CouponOffsetBusDayB>
<ResetOffsetBusDayB>true</ResetOffsetBusDayB>
<ResetHolidays>
    <Holiday>LON</Holiday>
</ResetHolidays>
<ResetTiming>BEG_PER</ResetTiming>
```

```
                    <CouponPaymentAtEnd>true</CouponPaymentAtEnd>
                </SwapLeg>
                <Discounted>false</Discounted>
            </InterestRateSwap>
        </Product>
     </CalypsoTrade>
</CalypsoUploadDocument>
```

Response Example

```
HTTP/1.1 200 OK
calcorrelationid: 703f63d7-1901-46fe-a0f7-008a17af69a6#junit/974bed94
Content-Type: application/json;charset=UTF-8
Content-Length: 641


<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CalypsoAcknowledgement UploadFile="IRS_Averaging.xml" UploadDate="Mon Aug 24 13:27:08 IDT 2020"
TotalTime="1407" Uploaded="1" Rejected="0" Received="1">
    <CalypsoTrades Uploaded="1" Rejected="0" Received="1">
        <CalypsoTrade>
            <ExternalRef>AveragingTrade_1</ExternalRef>
            <ProcessingOrg>GLOBAL BALANCED FUND</ProcessingOrg>
            <CalypsoTradeId>1135711</CalypsoTradeId>
            <Status>Success</Status>
            <Action>NEW</Action>
            <TradeCustomKeywords/>
        </CalypsoTrade>
    </CalypsoTrades>
</CalypsoAcknowledgement>
```

6.2.5 Upload Message as JSON

A `POST` uploads the data in JSON format. Returns Acknowledgement with status and error messages if any.

Request Parameters

Parameter	Description
persistMessage	optional: "Valid values are All,Failure,None. Default value is None."
ignoreWarnings	optional: "Valid values are true/false. Default value is true"

Request Example

```
POST /api/v2/datauploader/upload?persistMessage=None&ignoreWarnings=true HTTP/1.1
Content-Type: application/json
Host: localhost:8080
Content-Length: 6264

{
    "calypsoTrade": [
        {
            "externalReference": "S418996",
            "internalReference": "640931",
            "action": "NEW",
            "tradeCounterParty": "CME",
            "counterPartyRole": "CounterParty",
            "tradeBook": "Global",
            "tradeBundleOneMsg": false,
            "tradeEventsInSameBundle": false,
            "tradeCurrency": "EUR",
            "tradeSettlementCurrency": "USD",
            "negotiatedCurrency": "EUR",
            "tradeNotional": 100000.0,
            "buySell": "BUY",
            "tradeDateTime": "2020-04-17T13:21:00.000Z",
            "tradeSettleDate": "2020-04-21T00:00:00.000Z",
            "startDate": "2020-05-22T00:00:00.000Z",
            "maturityDate": "2020-05-19T00:00:00.000Z",
            "holidayCode": {},
```

```
            "productType": "FXOption",
            "product": {
                "fxoption": {
                    "optionStyle": "VANILLA",
                    "exerciseType": "European",
                    "primaryCurrency": "EUR",
                    "secondaryCurrency": "USD",
                    "optionType": "CALL",
                    "primaryAmount": 100000.0,
                    "quotingAmount": 27247.96,
                    "strikePrice": 0.27247956,
                    "settlementType": "Physical",
                    "settlementCurrency": "USD",
                    "quantoFactor": "NaN",
                    "expiryTimeZone": "Asia/Tel_Aviv",
                    "fxresetIsSplitB": false
                }
            },
            "tradeFee": {
                "fee": [
                    {
                        "feeType": "PREMIUM",
                        "feeDate": "2020-04-21T00:00:00.000Z",
                        "feeStartDate": "2020-04-21T00:00:00.000Z",
                        "feeEndDate": "2020-04-21T00:00:00.000Z",
                        "feeAmount": 3192.9,
                        "feeDirection": "PAY",
                        "feeCounterParty": "CME",
                        "currency": "EUR",
                        "manualAmountB": false,
                        "withOverrideB": false
                    },
                    {
                        "feeType": "COMMISSION",
                        "feeDate": "2020-04-21T00:00:00.000Z",
                        "feeStartDate": "2020-04-21T00:00:00.000Z",
                        "feeEndDate": "2020-04-21T00:00:00.000Z",
                        "feeAmount": 0.0,
```

```
                    "feeDirection": "PAY",
                    "feeCounterParty": "CME",
                    "currency": "USD",
                    "manualAmountB": false,
                    "withOverrideB": true
                }
            ]
        },
        "cashFlows": {},
        "tradeKeywords": {
            "keyword": [
                {
                    "keywordName": "DealModelPrice",
                    "keywordValue": "0.00000"
                },
                {
                    "keywordName": "RateSide",
                    "keywordValue": "Choice"
                },
                {
                    "keywordName": "TradeSource",
                    "keywordValue": "DataUploader"
                },
                {
                    "keywordName": "PSStrategyName",
                    "keywordValue": "Vanilla"
                },
                {
                    "keywordName": "ROUND_PRICE_TO",
                    "keywordValue": "0.25"
                },
                {
                    "keywordName": "RatesPrecision",
                    "keywordValue": "EUR/USD=8"
                },
                {
                    "keywordName": "ROUNDING",
                    "keywordValue": "NEAREST"
```

```
        },
        {
            "keywordName": "DealCcy1Rate",
            "keywordValue": "0.007"
        },
        {
            "keywordName": "DealFwdRate",
            "keywordValue": "0"
        },
        {
            "keywordName": "PremRate",
            "keywordValue": "Pips=0.00000000"
        },
        {
            "keywordName": "DealSpotDate",
            "keywordValue": "04-21-2020"
        },
        {
            "keywordName": "ExpiryDeliveryLink",
            "keywordValue": "Off"
        },
        {
            "keywordName": "CurrencyPair",
            "keywordValue": "EUR/USD"
        },
        {
            "keywordName": "DealPricingModel",
            "keywordValue": "FXOptionVanilla"
        },
        {
            "keywordName": "MarginFXRate",
            "keywordValue": "1"
        },
        {
            "keywordName": "PriceCcy",
            "keywordValue": "EUR"
        },
        {
```

```
                        "keywordName": "DealFwdPts",
                        "keywordValue": "0"
                    },
                    {
                        "keywordName": "DealModelPremium",
                        "keywordValue": "0"
                    },
                    {
                        "keywordName": "StrategyType",
                        "keywordValue": "Vanilla:640931,"
                    },
                    {
                        "keywordName": "NegotiatedCurrency",
                        "keywordValue": "EUR"
                    },
                    {
                        "keywordName": "DealCcy2Rate",
                        "keywordValue": "0.00561"
                    },
                    {
                        "keywordName": "ObjectId",
                        "keywordValue": "640931"
                    }
                ]
            },
            "marketPlace": "CME"
        }
    ],
    "version": "1"
}
```


Response Example


```
HTTP/1.1 200 OK
calcorrelationid: c02dddda-de64-4540-843e-53c643cad1d0#junit/2e88d98c
Content-Type: application/json;charset=UTF-8
Content-Length: 601
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CalypsoAcknowledgement UploadDate="Wed Sep 09 13:25:14 GMT 2020" TotalTime="8755" Uploaded="1" Rejected="0"
Received="1">
    <CalypsoTrades Uploaded="1" Rejected="0" Received="1">
        <CalypsoTrade>
            <ExternalRef>S418996</ExternalRef>
            <ProcessingOrg>GLOBAL BALANCED FUND</ProcessingOrg>
            <CalypsoTradeId>1135914</CalypsoTradeId>
            <Status>Success</Status>
            <Action>NEW</Action>
            <TradeCustomKeywords/>
        </CalypsoTrade>
    </CalypsoTrades>
</CalypsoAcknowledgement>
```

6.2.6 Upload Message as JSON and get Acknowledgement as JSON

A `POST` uploads the data in JSON format. Returns Acknowledgement in JSON format if in the request header 'Accept' is explicitly provided as 'application/json'.

Request Parameters

Parameter	Description
persistMessage	optional: "Valid values are All,Failure,None. Default value is None."
ignoreWarnings	optional: "Valid values are true/false. Default value is true"

Request Example

```
POST /api/v2/datauploader/upload?persistMessage=None&ignoreWarnings=true HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: localhost:8080
Content-Length: 6264

{
    "calypsoTrade": [
        {
            "externalReference": "S418996",
```

```
            "internalReference": "640931",
            "action": "NEW",
            "tradeCounterParty": "CME",
            "counterPartyRole": "CounterParty",
            "tradeBook": "Global",
            "tradeBundleOneMsg": false,
            "tradeEventsInSameBundle": false,
            "tradeCurrency": "EUR",
            "tradeSettlementCurrency": "USD",
            "negotiatedCurrency": "EUR",
            "tradeNotional": 100000.0,
            "buySell": "BUY",
            "tradeDateTime": "2020-04-17T13:21:00.000Z",
            "tradeSettleDate": "2020-04-21T00:00:00.000Z",
            "startDate": "2020-05-22T00:00:00.000Z",
            "maturityDate": "2020-05-19T00:00:00.000Z",
            "holidayCode": {},
            "productType": "FXOption",
            "product": {
                "fxoption": {
                    "optionStyle": "VANILLA",
                    "exerciseType": "European",
                    "primaryCurrency": "EUR",
                    "secondaryCurrency": "USD",
                    "optionType": "CALL",
                    "primaryAmount": 100000.0,
                    "quotingAmount": 27247.96,
                    "strikePrice": 0.27247956,
                    "settlementType": "Physical",
                    "settlementCurrency": "USD",
                    "quantoFactor": "NaN",
                    "expiryTimeZone": "Asia/Tel_Aviv",
                    "fxresetIsSplitB": false
                }
            },
            "tradeFee": {
                "fee": [
                    {
```

```
                    "feeType": "PREMIUM",
                    "feeDate": "2020-04-21T00:00:00.000Z",
                    "feeStartDate": "2020-04-21T00:00:00.000Z",
                    "feeEndDate": "2020-04-21T00:00:00.000Z",
                    "feeAmount": 3192.9,
                    "feeDirection": "PAY",
                    "feeCounterParty": "CME",
                    "currency": "EUR",
                    "manualAmountB": false,
                    "withOverrideB": false
                },
                {
                    "feeType": "COMMISSION",
                    "feeDate": "2020-04-21T00:00:00.000Z",
                    "feeStartDate": "2020-04-21T00:00:00.000Z",
                    "feeEndDate": "2020-04-21T00:00:00.000Z",
                    "feeAmount": 0.0,
                    "feeDirection": "PAY",
                    "feeCounterParty": "CME",
                    "currency": "USD",
                    "manualAmountB": false,
                    "withOverrideB": true
                }
            ]
        },
        "cashFlows": {},
        "tradeKeywords": {
            "keyword": [
                {
                    "keywordName": "DealModelPrice",
                    "keywordValue": "0.00000"
                },
                {
                    "keywordName": "RateSide",
                    "keywordValue": "Choice"
                },
                {
                    "keywordName": "TradeSource",
```

```
            "keywordValue": "DataUploader"
        },
        {
            "keywordName": "PSStrategyName",
            "keywordValue": "Vanilla"
        },
        {
            "keywordName": "ROUND_PRICE_TO",
            "keywordValue": "0.25"
        },
        {
            "keywordName": "RatesPrecision",
            "keywordValue": "EUR/USD=8"
        },
        {
            "keywordName": "ROUNDING",
            "keywordValue": "NEAREST"
        },
        {
            "keywordName": "DealCcy1Rate",
            "keywordValue": "0.007"
        },
        {
            "keywordName": "DealFwdRate",
            "keywordValue": "0"
        },
        {
            "keywordName": "PremRate",
            "keywordValue": "Pips=0.00000000"
        },
        {
            "keywordName": "DealSpotDate",
            "keywordValue": "04-21-2020"
        },
        {
            "keywordName": "ExpiryDeliveryLink",
            "keywordValue": "Off"
        },
```

```
{
    "keywordName": "CurrencyPair",
    "keywordValue": "EUR/USD"
},
{
    "keywordName": "DealPricingModel",
    "keywordValue": "FXOptionVanilla"
},
{
    "keywordName": "MarginFXRate",
    "keywordValue": "1"
},
{
    "keywordName": "PriceCcy",
    "keywordValue": "EUR"
},
{
    "keywordName": "DealFwdPts",
    "keywordValue": "0"
},
{
    "keywordName": "DealModelPremium",
    "keywordValue": "0"
},
{
    "keywordName": "StrategyType",
    "keywordValue": "Vanilla:640931,"
},
{
    "keywordName": "NegotiatedCurrency",
    "keywordValue": "EUR"
},
{
    "keywordName": "DealCcy2Rate",
    "keywordValue": "0.00561"
},
{
    "keywordName": "ObjectId",
```

```
                        "keywordValue": "640931"
                    }
                ]
            },
            "marketPlace": "CME"
        }
    ],
    "version": "1"
}
```

Response Example

```
HTTP/1.1 200 OK
calcorrelationid: c980aab2-4b3c-4189-b6b9-cc358ea64ded#junit/e4aceed3
Content-Type: application/json;charset=UTF-8
Content-Length: 326

{"calypsoTrades":{"uploaded":1,"rejected":0,"received":1,"calypsoTrade":[{"externalRef":"S418996","processingOrg
":"GLOBAL BALANCED
FUND","calypsoTradeId":1135914,"status":"Success","action":"NEW","tradeCustomKeywords":{}}]},"uploadDate":"Wed
Sep 09 13:25:14 GMT 2020","totalTime":"8755","uploaded":1,"rejected":0,"received":1}
```

6.2.7 Upload Message as XML in Asynch Mode

A `POST` uploads the data in XML format and it can contain multiple objects. Returns Acknowledgement with request id which should be used to get the acknowledgement and error messages if any.

Request Parameters

Parameter	Description
persistMessage	optional: "Valid values are All,Failure,None. Default value is None."
ignoreWarnings	optional: "Valid values are true/false. Default value is true"

Request Example

```
POST /api/v2/datauploader/upload-all?persistMessage=None&ignoreWarnings=true HTTP/1.1
Content-Type: application/xml
Host: localhost:8080
Content-Length: 8986

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<CalypsoUploadDocument>
    <CalypsoTrade>
        <ExternalReference>AveragingTrade_1</ExternalReference>
        <TradeId>0</TradeId>
        <Action>NEW</Action>
        <TradeCounterParty>CP</TradeCounterParty>
               <CounterPartyRole>CounterParty</CounterPartyRole>
        <TradeBook>Global</TradeBook>
        <TradeCurrency>USD</TradeCurrency>
        <TradeNotional>0.0</TradeNotional>
        <TradeDateTime>2012-05-11T18:38:48Z</TradeDateTime>
        <TradeSettleDate>2012-05-15</TradeSettleDate>
        <StartDate>2012-05-15</StartDate>
        <MaturityDate>2013-05-15</MaturityDate>
        <TraderName>trader20</TraderName>
        <SalesPerson>NONE</SalesPerson>
        <ProductType>InterestRateSwap</ProductType>
        <ProductSubType>BasisSwap</ProductSubType>
        <Product>
            <InterestRateSwap>
                <SwapLeg>
                    <LegType>Float</LegType>
                    <PayRec>REC</PayRec>
                    <Currency>USD</Currency>
                    <Amount>2.0E7</Amount>
                    <StartDate>2012-05-15</StartDate>
                    <EndDate>2013-05-15T23:59:00.000-04:00</EndDate>
                    <RateIndex>FEDFUNDS</RateIndex>
                    <RateIndexSource>FEDFUNDS1</RateIndexSource>
                    <Spread>0.5</Spread>
                    <Tenor>1D</Tenor>
                    <StubPeriod>NONE</StubPeriod>
                    <FirstReset>NONE</FirstReset>
                    <HolidayCode>
                        <Holiday>LON</Holiday>
                        <Holiday>NYC</Holiday>
                    </HolidayCode>
                    <InterestCompoundingFrequency>QTR</InterestCompoundingFrequency>
```

```
            <ResetFrequency>DLY</ResetFrequency>
            <ResetMethod>Weighted</ResetMethod>
            <FloatingRateReset>true</FloatingRateReset>
            <ResetLag>0</ResetLag>
            <PaymentFrequency>QTR</PaymentFrequency>
            <DateRollConvention>MOD_FOLLOW</DateRollConvention>
            <AccrualPeriodAdjustment>ADJUSTED</AccrualPeriodAdjustment>
            <RollDay>15</RollDay>
            <PaymentLag>0</PaymentLag>
            <DayCountConvention>ACT/360</DayCountConvention>
            <CouponOffsetBusDayB>true</CouponOffsetBusDayB>
            <ResetOffsetBusDayB>true</ResetOffsetBusDayB>
            <ResetHolidays>
                <Holiday>NYC</Holiday>
            </ResetHolidays>
            <CouponPaymentAtEnd>true</CouponPaymentAtEnd>
        </SwapLeg>
        <SwapLeg>
            <LegType>Float</LegType>
            <PayRec>PAY</PayRec>
            <Currency>USD</Currency>
            <Amount>2.0E7</Amount>
            <StartDate>2012-05-15</StartDate>
            <EndDate>2013-05-15T23:59:00.000-04:00</EndDate>
            <RateIndex>LIBOR</RateIndex>
            <RateIndexSource>LIBOR01</RateIndexSource>
            <Spread>0.2</Spread>
            <Tenor>3M</Tenor>
            <StubPeriod>NONE</StubPeriod>
            <FirstReset>NONE</FirstReset>
            <FirstRate/>
            <HolidayCode>
                <Holiday>LON</Holiday>
                <Holiday>NYC</Holiday>
            </HolidayCode>
            <InterestCompoundingFrequency>QTR</InterestCompoundingFrequency>
            <ResetLag>-2</ResetLag>
            <PaymentFrequency>QTR</PaymentFrequency>
```

```
                    <DateRollConvention>MOD_FOLLOW</DateRollConvention>
                    <AccrualPeriodAdjustment>ADJUSTED</AccrualPeriodAdjustment>
                    <RollDay>15</RollDay>
                    <PaymentLag>0</PaymentLag>
                    <DayCountConvention>ACT/360</DayCountConvention>
                    <CouponOffsetBusDayB>true</CouponOffsetBusDayB>
                    <ResetOffsetBusDayB>true</ResetOffsetBusDayB>
                    <ResetHolidays>
                        <Holiday>LON</Holiday>
                    </ResetHolidays>
                    <ResetTiming>BEG_PER</ResetTiming>
                    <CouponPaymentAtEnd>true</CouponPaymentAtEnd>
                </SwapLeg>
                <Discounted>false</Discounted>
            </InterestRateSwap>
        </Product>
    </CalypsoTrade>
<CalypsoTrade>
        <ExternalReference>AveragingTrade_2</ExternalReference>
        <TradeId>0</TradeId>
        <Action>NEW</Action>
        <TradeCounterParty>CP</TradeCounterParty>
               <CounterPartyRole>CounterParty</CounterPartyRole>
        <TradeBook>Global</TradeBook>
        <TradeCurrency>USD</TradeCurrency>
        <TradeNotional>0.0</TradeNotional>
        <TradeDateTime>2012-05-11T18:38:48Z</TradeDateTime>
        <TradeSettleDate>2012-05-15</TradeSettleDate>
        <StartDate>2012-05-15</StartDate>
        <MaturityDate>2013-05-15</MaturityDate>
        <TraderName>trader20</TraderName>
        <SalesPerson>NONE</SalesPerson>
        <ProductType>InterestRateSwap</ProductType>
        <ProductSubType>BasisSwap</ProductSubType>
        <Product>
            <InterestRateSwap>
                <SwapLeg>
                    <LegType>Float</LegType>
```

```
            <PayRec>REC</PayRec>
            <Currency>USD</Currency>
            <Amount>2.0E7</Amount>
            <StartDate>2012-05-15</StartDate>
            <EndDate>2013-05-15T23:59:00.000-04:00</EndDate>
            <RateIndex>FEDFUNDS</RateIndex>
            <RateIndexSource>FEDFUNDS1</RateIndexSource>
            <Spread>0.5</Spread>
            <Tenor>1D</Tenor>
            <StubPeriod>NONE</StubPeriod>
            <FirstReset>NONE</FirstReset>
            <HolidayCode>
                <Holiday>LON</Holiday>
                <Holiday>NYC</Holiday>
            </HolidayCode>
            <InterestCompoundingFrequency>QTR</InterestCompoundingFrequency>
            <ResetFrequency>DLY</ResetFrequency>
            <ResetMethod>Weighted</ResetMethod>
            <FloatingRateReset>true</FloatingRateReset>
            <ResetLag>0</ResetLag>
            <PaymentFrequency>QTR</PaymentFrequency>
            <DateRollConvention>MOD_FOLLOW</DateRollConvention>
            <AccrualPeriodAdjustment>ADJUSTED</AccrualPeriodAdjustment>
            <RollDay>15</RollDay>
            <PaymentLag>0</PaymentLag>
            <DayCountConvention>ACT/360</DayCountConvention>
            <CouponOffsetBusDayB>true</CouponOffsetBusDayB>
            <ResetOffsetBusDayB>true</ResetOffsetBusDayB>
            <ResetHolidays>
                <Holiday>NYC</Holiday>
            </ResetHolidays>
            <CouponPaymentAtEnd>true</CouponPaymentAtEnd>
        </SwapLeg>
        <SwapLeg>
            <LegType>Float</LegType>
            <PayRec>PAY</PayRec>
            <Currency>USD</Currency>
            <Amount>2.0E7</Amount>
```

```
                    <StartDate>2012-05-15</StartDate>
                    <EndDate>2013-05-15T23:59:00.000-04:00</EndDate>
                    <RateIndex>LIBOR</RateIndex>
                    <RateIndexSource>LIBOR01</RateIndexSource>
                    <Spread>0.2</Spread>
                    <Tenor>3M</Tenor>
                    <StubPeriod>NONE</StubPeriod>
                    <FirstReset>NONE</FirstReset>
                    <FirstRate/>
                    <HolidayCode>
                        <Holiday>LON</Holiday>
                        <Holiday>NYC</Holiday>
                    </HolidayCode>
                    <InterestCompoundingFrequency>QTR</InterestCompoundingFrequency>
                    <ResetLag>-2</ResetLag>
                    <PaymentFrequency>QTR</PaymentFrequency>
                    <DateRollConvention>MOD_FOLLOW</DateRollConvention>
                    <AccrualPeriodAdjustment>ADJUSTED</AccrualPeriodAdjustment>
                    <RollDay>15</RollDay>
                    <PaymentLag>0</PaymentLag>
                    <DayCountConvention>ACT/360</DayCountConvention>
                    <CouponOffsetBusDayB>true</CouponOffsetBusDayB>
                    <ResetOffsetBusDayB>true</ResetOffsetBusDayB>
                    <ResetHolidays>
                        <Holiday>LON</Holiday>
                    </ResetHolidays>
                    <ResetTiming>BEG_PER</ResetTiming>
                    <CouponPaymentAtEnd>true</CouponPaymentAtEnd>
                </SwapLeg>
                <Discounted>false</Discounted>
            </InterestRateSwap>
        </Product>
    </CalypsoTrade>
</CalypsoUploadDocument>
```

Response Example

```
HTTP/1.1 200 OK
```

```
calcorrelationid: c5565fe3-e3cc-4625-859b-634a474c7065#junit/0d45a488
Content-Type: application/json;charset=UTF-8
Content-Length: 357

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CalypsoAcknowledgement Uploaded="0" Rejected="0" Received="0">
    <WebServiceResponse>
        <RequestId>480502</RequestId>
        <Status>Processing</Status>
        <Description>Request has been accepted and ready for Processing</Description>
    </WebServiceResponse>
</CalypsoAcknowledgement>
```

6.2.8 Fetching Acknowledgement using Request id

A `GET` returns the Acknowledgement for the message that was uploaded in asynch mode using requestId

Request Parameters

Parameter	Description
requestId	"Request id that was generated when upload was done in asynch mode using the end point upload-all."

Request Example

```
GET /api/v2/datauploader/acknowledgement?requestId=480502 HTTP/1.1
Content-Type: application/json
Host: localhost:8080
```

Response Example

```
HTTP/1.1 200 OK
calcorrelationid: 8c95d035-3898-4bbe-bf0a-aec93ad4e1d7#junit/843049e4
Content-Type: application/json;charset=UTF-8
Content-Length: 933

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CalypsoAcknowledgement UploadDate="Thu Sep 10 14:10:29 GMT 2020" TotalTime="85" Uploaded="2" Rejected="0"
Received="2">
```

```
    <CalypsoTrades Uploaded="2" Rejected="0" Received="2">
        <CalypsoTrade>
            <ExternalRef>AveragingTrade_1</ExternalRef>
            <ProcessingOrg>GLOBAL BALANCED FUND</ProcessingOrg>
            <CalypsoTradeId>1136030</CalypsoTradeId>
            <Status>Success</Status>
            <Action>NEW</Action>
            <TradeCustomKeywords/>
        </CalypsoTrade>
        <CalypsoTrade>
            <ExternalRef>AveragingTrade_2</ExternalRef>
            <ProcessingOrg>GLOBAL BALANCED FUND</ProcessingOrg>
            <CalypsoTradeId>1136029</CalypsoTradeId>
            <Status>Success</Status>
            <Action>NEW</Action>
            <TradeCustomKeywords/>
        </CalypsoTrade>
    </CalypsoTrades>
</CalypsoAcknowledgement>
```

6.3 Sample Client Program to call the REST End Points

```
package com.calypso.tk.util;


import java.io.File;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.util.Base64;
import org.apache.commons.io.FileUtils;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestTemplate;
import com.calypso.tk.core.Log;
```

```
public class TestClientWebService{


	private static final String UPLOADER = "UPLOADER";
	static HttpHeaders headers = new HttpHeaders();
	static HttpEntity<Object> entity = null;


	static{
		//setting accept
		//headers.setAccept(Arrays.asList(MediaType.APPLICATION_XML,MediaType.APPLICATION_JSON));


		//Setting content type
		headers.setContentType(MediaType.TEXT_PLAIN);


		//setting authorization as basic authorization
		String auth = "calypso_user:calypso";
		byte[] encodedAuth = Base64.getEncoder().encode(auth.getBytes(Charset.forName("US-ASCII")));
		String authHeader = "Basic " + new String(encodedAuth);
		headers.set("Authorization", authHeader);


		try {
			//Reading the CalypsoUploadDocument xml data from a file to upload.
			String body = FileUtils.readFileToString(new
File("C:\\calypso\\datauploader\\samples\\IRS.xml"),
					StandardCharsets.UTF_8);
			entity = new HttpEntity<Object>(body, headers);
		} catch (IOException e) {
			Log.error(UPLOADER, e);
		}
	}


	public static <T>ResponseEntity<String> invokeRest(String restEndPoint, HttpMethod method, HttpEntity<T>
entity){
		return new RestTemplate().exchange(restEndPoint, method, entity, String.class);
	}


	public static void main(String[] args) {
		//Calling rest end point to upload the data
		//"http://<IP>:<port of calypso services>/datauploader-service/api/v2/datauploader/upload-
message"
```

```
		String restEndPoint = "http://localhost:9140/datauploader-service/api/v2/datauploader/upload-
message";


		if (entity!= null){
			ResponseEntity<String> response=invokeRest(restEndPoint,HttpMethod.POST, entity);


			System.out.println( response.getHeaders().toString());
			System.out.println( response.getStatusCode().toString());
			System.out.println(response.getBody().toString());


		}
	}
}
```

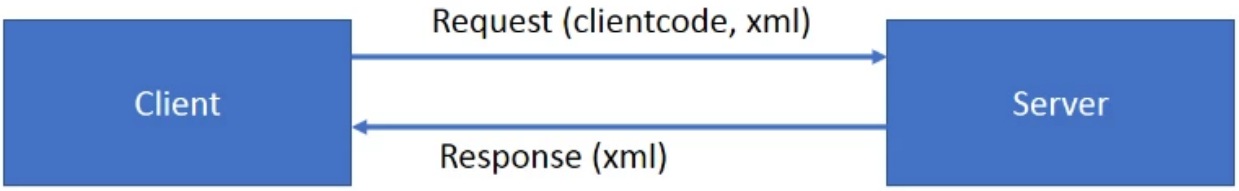
6.4 Service Type

Data Uploader web-services allow user to upload data in the following ways.

a. Synchronous
b. Asynchronous

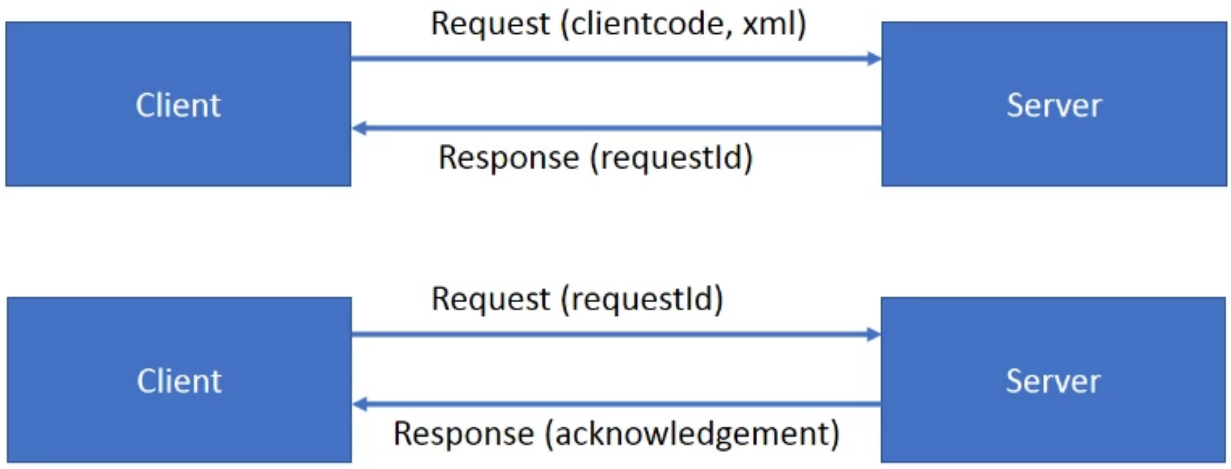
A. Synchronous

In case of synchronous service, client request is blocked, till the message(s) are processed and uploaded. After successfully processing the message, the Calypso Acknowledgement will be returned. As this blocks the client's call, the number of objects to be uploaded via this service should be minimal. So, this approach should be used for single objects.



B. Asynchronous

This service is for bulk upload. Here, client request for upload service is not block. When client request for upload of an upload, the API will immediately return request-id. The API then publishes an event/message to Update Manager Engine which uploads all the objects in asynchronous fashion, and store the result in the database, which the client can request by passing the request-id given before.



Requirement for asynchronous API

i. Import UPLOADREQUESTMSG and UPLOADRESPONSEMSG workflow

ii. UpdateManagerEngine must be correctly configured and running.

Limitation:

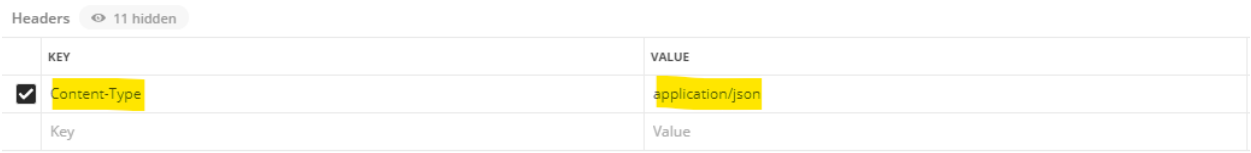
There is a limitation of 10Mb on the HTTP post data to be exchanged, around 5k trades can be accommodated within 10Mb size.

Archival / Cleanup

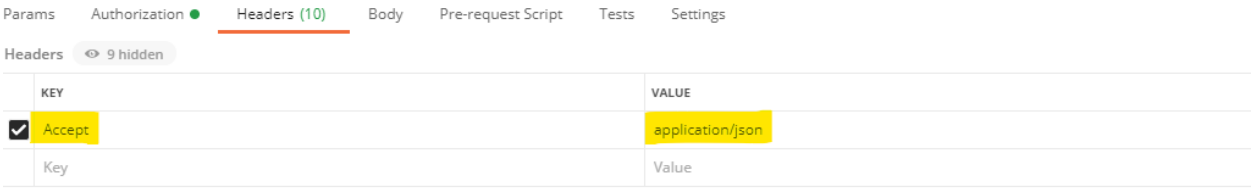
The Messages that are created can be archived/purged via the Archival Scheduled Tasks that Calypso provides.

For uploading data in json format

Here in the header Content-Type should be set to application/json as shown below



Similarly, for getting trades from **/trades** API, user can pass the header as shown below to retrieve the response in json format.


Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
Headers
9 hidden
KEY
VALUE
Accept
application/json
Key
Value

Using the Local Mode with Persistent Messages

[NOTE: This feature is only applicable from Data Uploader version 4.0.0 onwards.]

In order to improve performance of the Data Uploader, the framework has been changed to execute the upload process in the client side instead of the data server via the workflow rules. Workflow rules are executed in the data server thus keeping data server busy every time data is uploaded. Client side execution has the advantage of using the cache, and can thus perform better.

This change provides all the features that were available via the workflow based approach such as,

Persisting external messages a BO Messages.

Ability to Re-process failed messages.

Maintains the order in which the messages are received

Acknowledgement generation.

The framework makes use of the following configuration properties (in the property files `datauploader.properties` and `calypso_uploader_config.properties`).

uploadMode:- Set to "Local". Local is for using the API (Local API).

persistMessages:- Possible values are 'All', 'None' and 'Failure'

- All - External Messages are always persisted as BO Messages.
- None - External Messages are not persisted, which means that if the message fails in translation or validation, the message needs to be resent.
- Failure - External Messages are persisted only in case of failure (when translation or validation fails). This enables the failed messages to be reprocessed.

Only the pending Messages created via the local mode, are Re-processed via the Update Manager engine, which runs in the engine server.

The message workflows are changed to generate an event UploadReprocess, every time a failed message is re-processed. The Update Manager engine receives these events and processes them again and generates acknowledgement if needed.

The Update Manager engine needs to subscribe to the following events:

PSEventUploadReprocess

PSEventRepublish

Please refer to the Calypso Data Uploader Integration guide for complete details.

7.1 How to use Local Mode in API

The following is the sample code that explains how to get these features from an API.

The API is present in the file "`DataUploadMessageHandler.java`".

The method is uploadMessage(IUploadMessage message) which takes an instance of IUploadMessage and returns a CalypsoAcknowledgement object.

```
public CalypsoAcknowledgement uploadMessage(IUploadMessage uploadMessage) throws Exception


GatewayMessage gatewaymessage = new GatewayMessage (uploaderxml, "UploaderXML");
MessageInfo info = new MessageInfo();
info.setAttribute("uploadMode", "Local");
info.setAttribute("persistMessages", "All");


/*
All indicates persist all the messages.
Failure indicates persist only failed messages
*/
gatewaymessage.setMessageInfo(info);


DataUploadMessageHandler handler = new DataUploadMessageHandler ();
CalypsoAcknowledgement ack = handler.uploadMessage(gatewaymessage);
```

7.2 How to use Local Mode in File Watcher or Import Message Engine

The File watcher and the Import Message Engine read the uploadMode and persistMessages attributes from the configuration files "`datauploader.properties`" and "`calypso_uploader_config.properties`" respectively.

So, update the properties in the respective files and the data uploader framework will make use of these properties and upload the external messages.

Extensions

8.1 How to Write a Customizer

A Customizer provides a way for the user to customize the payload which is the JAXB POJO, based on business needs before the actual Validator / Uploader are called.

The customizer is called from the workflow rules, based on the message source as follows:

`com.calypso.tk.upload.customizer.<MessageSource>UploadCustomizer.java`

The source can be specified in the Root tag of the XML as an attribute as shown below.

```
<CalypsoUploadDocument Source="sourcename">
```

Examples:

- In normal usage for Data uploader, (when uploading XML / CSV files) the source is defaulted to "UploaderXML".

  The customizer class = `com.calypso.tk.upload.customizer.UploaderXMLUploadCustomizer.java`

- And in case of Markitwire the source is "MW"

  The customizer class = `com.calypso.tk.upload.customizer.MWUploadCustomizer.java`

All the customizer classes should be derived from `com.calypso.tk.upload.customizer.DefaultUploadCustomizer.java`; however if Calypso provides any source based customizer (which can be found in the package `com.calypso.tk.upload.customizer`), the user has to derive from that Source based customizer and not the DefaultUploadCustomizer.

Sample customizer class:

```
package calypsox.tk.upload.customizer;


import java.util.Vector;


import com.calypso.tk.bo.BOException;
import com.calypso.tk.upload.customizer.DefaultUploadCustomizer;
import com.calypso.tk.upload.jaxb.CalypsoObject;


public class UploaderXMLUploadCustomizer extends DefaultUploadCustomizer {


	public void validate(CalypsoObject calobj,Object obj,Vector<BOException> error,Object dbCon) {
```

```
                //TODO
                //Check eligibility criteria
                //populate errors collection as follows

                //there will be util methods available in ErrorExceptionsUtil.java
                //for example if you do not want to trade on mexican currency

                CalypsoTrade calTrade = (CalypsoTrade) object;
                if (calTrade.getTradeCurrency().equals("MXN")) {
                        errors.add(ErrorExceptionUtils.createRejection("Currency", "100000",
calTrade.getTradeCurrency()));

//The 3 Parameters are explained as follows:
//1) The Field, in the JAXB Object that caused the rejection
//2) The Actual User defined Error Message (defined in ErrorCodeBundle.properties)
//100000 is defined in ErrorCodeBundle.properties as below
//100000=Invalid Currency for Clearing Trades
//3) The Value to be displayed along with error message
                }
        }

        public void load(CalypsoObject calobj,Object obj,Vector<BOException> error,Object dbCon) {
                //TODO
                //Check eligibility criteria
                //returning false means message will be rejected
        }
}
```

[NOTE: If there are multiple sources to handle the interface, there would be multiple Source based Upload Customizer classes, one for each source, and there might be a situation where all the interfaces would need to do some common processing, which can be written in a common class. For this purpose, you can use `CustomUploadCustomizer.java`; Calypso does not provide this class. If required the user has to write the common code in this class and place this class in the CLASSPATH, using the derivation mechanism explained above.]

The workflow rule would then call the Source based customizer if present, in the CLASSPATH, or call CustomUploadCustomizer, if present.

If a Source based customizer is provided, CustomUploadCustomizer one will never be called.

8.2 How to Extend an Existing Validator / Uploader

All the Validator classes are derived from an abstract class and implement the following method.

```
public abstract void validate(CalypsoObject object, Vector<BOException> errors);


/*Core functionality to validate the fields in the JAXB object is written in this method.*/


and similarly all the Uploader classes are derived from a base class and implement the following method.


public abstract void upload (CalypsoObject object,Vector<BOException> errors);


/*Core functionality to construct the object from the JAXB, is written in this method.*/
```

This method is implemented to construct the object being uploaded via the Data Uploader from the JAXB object, and is then saved in a separate method, saveObject(), thus providing ability for the implementor to change the object before the object is getting saved.

These methods can be used as extension points and can be overwritten in custom classes based on business needs.

For, example let us consider the class "`com.calypso.tk.upload.validator.ValidateCalypsoBook.java`", which is already written and you have a specific business requirement to do some validation. In this case, create a custom class named "`tk.upload.validator.ValidateCalypsoBook.java`" that extends "`com.calypso.tk.upload.validator.ValidateCalypsoBook.java`" as follows:

```
package calypsox.tk.upload.validator;


import java.util.Vector;


import com.calypso.tk.bo.BOException;
import com.calypso.tk.upload.jaxb.CalypsoObject;


public class ValidateCalypsoBook extends            com.calypso.tk.upload.validator.ValidateCalypsoBook {


       public void validate(CalypsoObject object, Vector<BOException> errors)
       {
              // do some pre processing (you can change the JAXB Object to meet the core validate()
functionality


              //call super.validate
              super.validate(object, errors);
```

```
                // do some post processing
        }
}
```

and a custom class named "`tk.upload.uploader.UploadCalypsoBook.java`" extends "`com.calypso.tk.upload.uploader.UploadCalypsoBook.java`" as follows:

```
package calypsox.tk.upload.uploader;


import java.util.Vector;


import com.calypso.tk.bo.BOException;
import com.calypso.tk.upload.jaxb.CalypsoObject;


public class UploadCalypsoBook extends              com.calypso.tk.upload.uploader.UploadCalypsoBook {


        public void upload(CalypsoObject object, Vector<BOException> errors)
        {
                //do some pre processing logic


                //call super.upload()
                super.upload(object, errors);


                //get the object that is being uploaded by calling getUploadObject() and type cast it
accordingly.
                ((Book)getUploadObject()).setAttribute("ProfitCenter", "Test123");
        }


}
```

Sample Validator Extension for Trade:

```
package calypsox.tk.upload.validator;


import java.util.Vector;


import com.calypso.tk.bo.BOException;
import com.calypso.tk.upload.jaxb.CalypsoObject;


public class ValidateCalypsoTradeInterestRateSwap extends
```

```
                com.calypso.tk.upload.validator.ValidateCalypsoTradeInterestRateSwap {


        public void validate(CalypsoObject object, Vector<BOException> errors) {
                //pre processing
                System.out
                                .println("My Extension ValidateCalypsoTradeInterestRateSwap.validateObject()");
                super.validate(object, errors);
                //post processing
        }
}
```

Sample Uploader Extension for Trade:

```
package calypsox.tk.upload.uploader;


import java.util.Vector;


import com.calypso.tk.bo.BOException;
import com.calypso.tk.upload.jaxb.CalypsoObject;


public class UploadCalypsoTradeInterestRateSwap extends
                com.calypso.tk.upload.uploader.UploadCalypsoTradeInterestRateSwap {


        public void upload(CalypsoObject object, Vector<BOException> errors) {
                //call super.upload first so that the trade object is constructed
                //then the trade object can be changed according to requirement
                super.upload(object, errors);


                //post processing
                trade.addKeyword("AccountNumber", "123456aaa");
        }
}
```

At runtime, the framework will make sure that if a class exists in a custom package, it is called by default; otherwise the core "com.calypso" class will be called.

When using the extension mechanisms explained above, depending on the requirements, the following utility methods can be used to create ERROR vs REJECT vs WARNING exceptions, which result in a Task Station entry with different types.

```
ErrorExceptionUtils.createRejection("Currency", "50000000 ", trade.getTradeCurrency());
```

```
ErrorExceptionUtils.createException("Trade Date", "50000001 ", trade.getTradeDate());

ErrorExceptionUtils.createWarning("Counterparty", "50000002 ", trade.getTradeCounterparty());
```

Please note that, when createRejection() and createException() are called, the message is blocked, and depending on the configuration, the rejected message can be moved to a different status. All the warnings are ignored.

8.3 Access the Error Code from the Extension Point

This section describes how to access the error codes, if the super class has created any.

Example:

```
public class ValidateCalypsoBook extends            com.calypso.tk.upload.validator.ValidateCalypsoBook {


       public void validate(CalypsoObject object, Vector<BOException> errors)
       {
              // do some pre processing (you can change the JAXB Object to meet the core validate()
functionality


              //call super.validate
              super.validate(object, errors);


              //errors collection contains the errors caused the validator of the core //uploader class, in
order to access the error code you need to do the following.


              if (!Util.isEmpty(errors)) {
              {
                     Iterator iter = errors.iterator();
                     while(iter.hasNext()) {
                     UploadBOException boException = (UploadBOException) iter.next();
                     String typeCode = boException.getTypeCode();
                     String msgCode = boException.getMsgCode();
                     String errormessage = boException.getMessage();


              //typeCode denotes the code for invalid data or missing data
              //msgCode denotes the code for the actual error message
                     }
```

```
                }
        }
}
```

8.4 How to handle Rejections

In the Data Uploader framework, there is way to reject messages based on some validation that is written by users without actually calling the Validator component; currently there is a validation extension, using the custom extension mechanism, which is object specific, and when the validation fails the message is sent back to the previous status, and stays there until either the message is cancelled or corrected (if any mapping is missing).

The following extension mechanism allows users to write a class deriving from the Calypso provided base class or interface and implement a method.

Currently there are 2 entry points into the framework

- From the Validate Message Rule calls the
  - call customizer object
  - call validator object
- From the Loader Message Rule calls the
  - call customizer object
  - call loader object

In the current framework, the customizer has the following methods,

```
        /**
         * This method is called in validate, you can write custom functionality here
         * @param obj
         * @param trade
         * @param error
         */

public void validate(CalypsoObject calobj,Object obj,Vector<BOException> errors);

        /**
         * This method is called in load message rule, you can write custom functionality here
         * @param obj
         * @param trade
         * @param error
         */
public void load(CalypsoObject calobj,Object obj,Vector<BOException> errors);
```

Which are used to customize the object before proceeding to the Validator or Loader component (that is modify the JAXB object based on business needs, which is persisted before moving further in the workflow).

The proposal is to use these as extension points by implementing them in the custom extension classes.

These methods take as parameters a collection of BO Exception objects (Vector<BOException> errors). The custom code, based on the business requirements, populates the BO Exception into the collection by calling the appropriate Utility methods that will mark the BO Exception category as REJECTION.

Please see sample code explained below. It is important to use the Correct API to create BOException which indicates a **REJECTION**.

Once the custom code is executed, the framework scans the collection, and monitors if any of the BO Exception is marked for REJECTION:

- **If true**, the message attributes (specified below) are added to the message, and the rule **returns false**, so that the static data filter will move the message to the designated status.
- **Otherwise** the workflow proceeds to the next transition.

Message Attributes:

- UploadObjectStatus - Rejected
- UploadObjectReason - <actual reason>

[Note: The above explanation applies to Validator Extensions as well.]

Sample customizer class:

```
package calypsox.tk.upload.customizer;


import java.util.Vector;


import com.calypso.tk.bo.BOException;
import com.calypso.tk.upload.jaxb.CalypsoObject;


public class UploaderXMLUploadCustomizer extends DefaultUploadCustomizer {


	public void validate(CalypsoObject calobj,Object obj,Vector<BOException> error) {
		//TODO
		//Check eligibility criteria
		//populate errors collection as follows
```

```
                //there will be util methods available in ErrorExceptionsUtil.java
                //for example if you do not want to trade on mexican currency

                CalypsoTrade calTrade = (CalypsoTrade) object;
                if (calTrade.getTradeCurrency().equals("MXN")) {
                        errors.add(ErrorExceptionUtils.createRejection("Currency", "200000",
calTrade.getTradeCurrency()));

//The 3 Parameters are explained as follows:
//1) The Field, in the JAXB Object that caused the rejection
//2) The Actual User defined Error Message (defined in ErrorCodeBundle.properties)
//100000 is defined in ErrorCodeBundle.properties as below
//100000=Invalid Currency for Clearing Trades
//3) The Value to be displayed along with error message
                }
        }

        public void load(CalypsoObject calobj,Object obj,Vector<BOException> error) {
                //TODO
                //Check eligibility criteria
                //returning false means message will be rejected
        }
}
```

8.5 Error Codes in Extension Mechanism

When using the extension mechanisms explained in the sections above, the extension framework allows defining error codes and allows using them.

All the error codes for the core functionality are defined in the file "`ErrorCodeBundle.properties`".

It is recommended to create a file named "`CustomErrorCodeBundle.properties`" for user defined error codes so that there will not be issues with merging the properties file in upgrades. However, please note that the error codes should start with a higher number like 50000000 onwards to avoid any discrepancies between core error codes and the user defined ones.

Examples:

```
ErrorExceptionUtils.createRejection("Currency", "50000000 ", trade.getTradeCurrency());
```

```
ErrorExceptionUtils.createException("Trade Date", "50000001", trade.getTradeDate());

ErrorExceptionUtils.createWarning("Counterparty", "50000002 ", trade.getTradeCounterparty());
```

CustomErrorCodeBundle.properties:

```
50000000=Invalid Currency for Trading

50000001=Trade Date cannot be after Maturity Date

50000002=Invalid Counterparty
```

8.6 Writing Translator Extension to Handle Rejections

All the translator classes are present in `com.calypso.tk.upload.translator`. Any extension to this class should be present in `tk.upload.translator`.

Consider the case where the ClearingSWMLTranslator needs to be extended to handle Rejections.

A class with the same name should be created in the custom package under `tk.upload.translator` and the translate() method should be overwritten as shown below.

```
package calypsox.tk.upload.translator;

public class ClearingSWMLTranslator extends com.calypso.tk.upload.translator.ClearingSWMLTranslator {

        public CalypsoUploadDocument translate(IUploadMessage uploadMessage, Vector<BOException> exceptions) {


/*call super class to perform the translation*/

CalypsoUploadDocument document = super.translate(uploadMessage, exceptions);


/*purpose of this extension is to reject the message if the counterparty bic code is not mapped. In this case
the base translator will validate the counterparty and create an exception in the collection exceptions, the
second parameter in the method signature.

The user now have to iterate this collection to see if the error related to the counterparty is present, and if
present change the exception type from '_ERROR' to '_REJECT'

Assume the error code is 10001 then the code would be as follows


NOTE: The exceptions returned by the base class can contain warnings as well with exception type as
EX_UPLOADSOURCEMSG_WARNING. These exceptions should not be modified at all. Only the ERROR's should be converted
to REJECT to REJECT the message.

*/

                Iterator<BOException> iter = exceptions.iterator();

                While(iter.hasNext()) {

                        BOException excep = iter.next();

                        if (excep instance of UploadBOException) {

                                UploadBOException uploadException = (UploadBOException) excep;
```

```
                                If (uploadException.getMsgCode().equals("10001") {
                                        //Change the exception type of the bo exception from _ERROR to _REJECT
                                }
                        }
                }
        }
}
```

Once this is done, the Translate Message Rule would iterate to see if there are any rejections available in the exceptions collection, and if any REJECT is present, the rule would add the Rejection attributes on the message (UploadObjectStatus=Rejected), and the message would then be sent to REJECTED status via the static data filter.

8.7 Important Domain Values

Define the following domain values to support Trade Workflow rules.

Domain UploadTerminateAction:

This domain holds the action to apply when a trade is terminated. If this value is not defined, the default is TERMINATE.

Domain UploadNovateAction:

This is the workflow action to apply when a trade is novated. If this value is not defined, the default is TERMINATE.

Domain UploadExitAction:

This is the workflow action to apply when you must exit from an outside trade (e.g. a MarkitWire trade). This action will affect the upstream trade outside Calypso.

Domain UploadAmendAction:

This domain holds the value, which is used as an action whenever an object needs to be amended, for example in rules, or based on business need.

Domain UploadTerminationReason:

This provides the default reason for a trade termination. The reason is stored as a keyword (TerminationReason) on the trade. If this value is not defined, the default is assigned.

Domain UploadNovationReason:

This provides the default reason for a partial trade termination. The reason is stored as a (NovationReason) keyword on the trade. If this value is not defined, the default is assigned.

Domain UploadAllowedAmendActions:

This domain lists the user-defined amend actions. The default action used for amendments is AMEND, however, when using the Data Uploader, that action can be changed to something else. If some other action must be used for amendments in XML/CSV upload, add that action in this domain and to the UploadAmendAction domain as well.

Domain UploadDeClearAction

Domain UploadRejectAction:

This domain holds the action to apply when doing Undo termination.

Domain UploadExerciseAction:

This domain holds the action to apply when a trade is exercised. If this value is not defined, the default is EXERCISE.

Performance Tuning

9.1 Database Indexes

The following two indexes should be applied to the database (please adapt the exact SQL required based on the specific RDBMS you are using, and change the tablespace name accordingly):

```
create index idx_trade_eref on trade (EXTERNAL_REFERENCE) tablespace calypso compute statistics;

create index idx_msg_attr_name_value on mess_attributes (ATTR_NAME, ATTR_VALUE) tablespace calypso compute statistics;
```

Please review the tablespaces and index names and adjust as per your database.

[Note: These may already be defaulted in newer Calypso versions.]

9.2 File Watcher / Uploader Import Message Engine Mode

To run the File Watcher or the Uploader Import Message engine in their different modes, update the property – uploadMode – in the file "`datauploader.properties`" (used by the File Watcher), or the file "`calypso_uploader_config.properties`" (used by the Uploader Import Message engine).

Please set the properties as below.

```
uploadMode=Local

persistMessages=Failure/None // improves performance as the BO Message creation is not there - only for failed messages
```

9.3 File Watcher Threads

To change the maximum number of threads used by the File Watcher when uploading files containing multiple trades, set the ThreadPoolSize property in the file "`gatewayservices.properties`".

```
ThreadPoolSize=4
```

Please set the pool size to match the number of CPU cores on the machine, if possible. In windows it will sometimes show two CPUs per core if they are hyper-threaded, but for this purpose it is actually one.

9.4 Engine Server

Calypso recommends adding the integration engines like Data Uploader engine and Uploader Import Message engine in a separate Engine Server to separate them from the Calypso back office engines.

This provides ability to isolate the integration components and avoid any fallout from the integration layer to the standard Calypso processing.

9.5 Uploader XML

There are several ways to improve your uploader XML to ensure it can be efficiently processed by the Data Uploader framework.

Below is a list of some items to be aware of, but as always, reading the Data Uploader documentation for the specific product / trade types you are uploading, is the best way to ensure your XML is formed as efficiently as possible.

- Do not include a ProductSubType value when none is required (e.g. Futures).

Review the documentation for the trade type you are uploading to see if it supports providing a product code/value pair for its underlying. If you are uploading those product underlyings as well, simply adding a unique product code/value pair to the product XML, and then referring to that within the trade XML, can significantly speed up lookups within the uploaders.